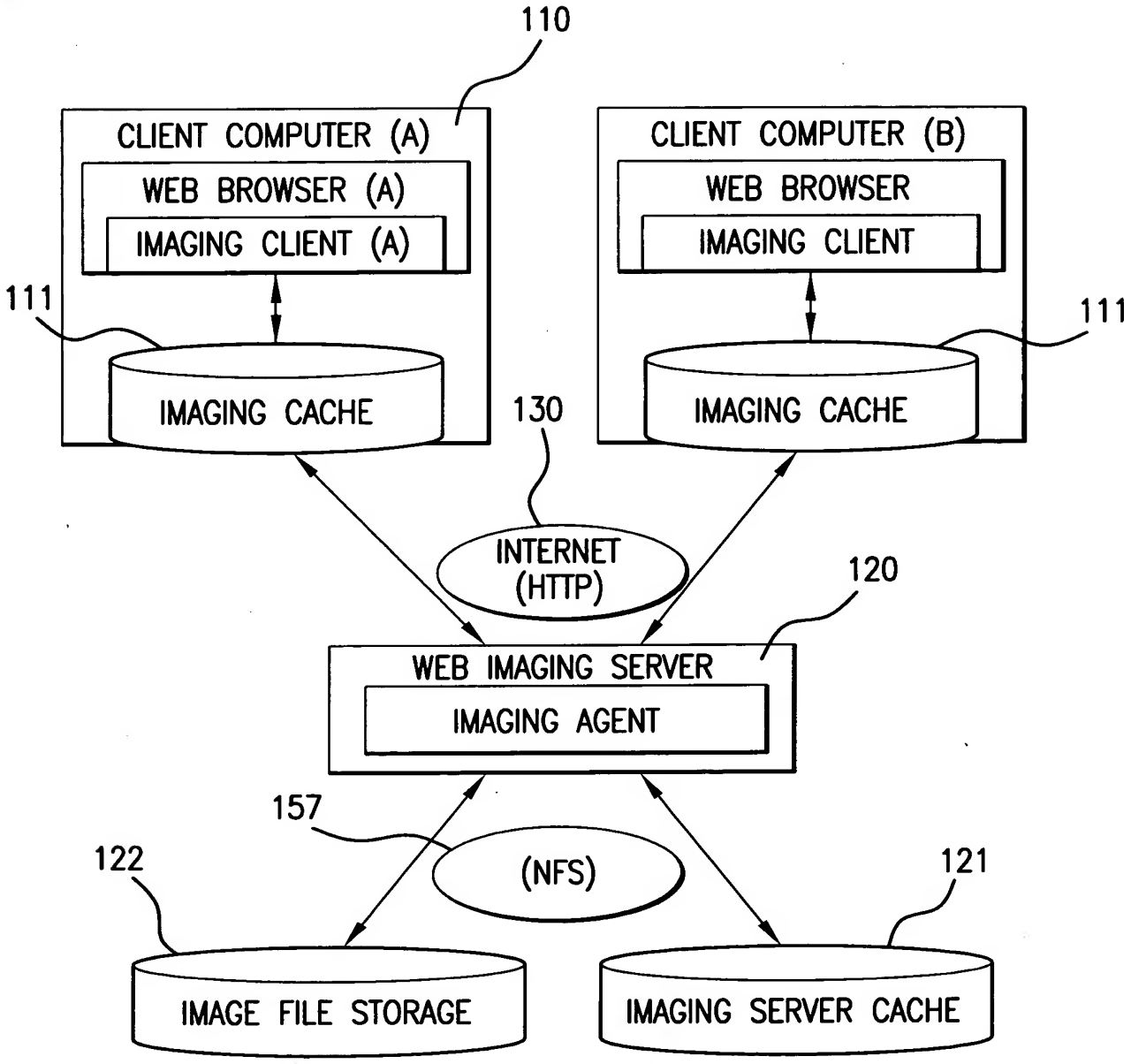


FIG.1

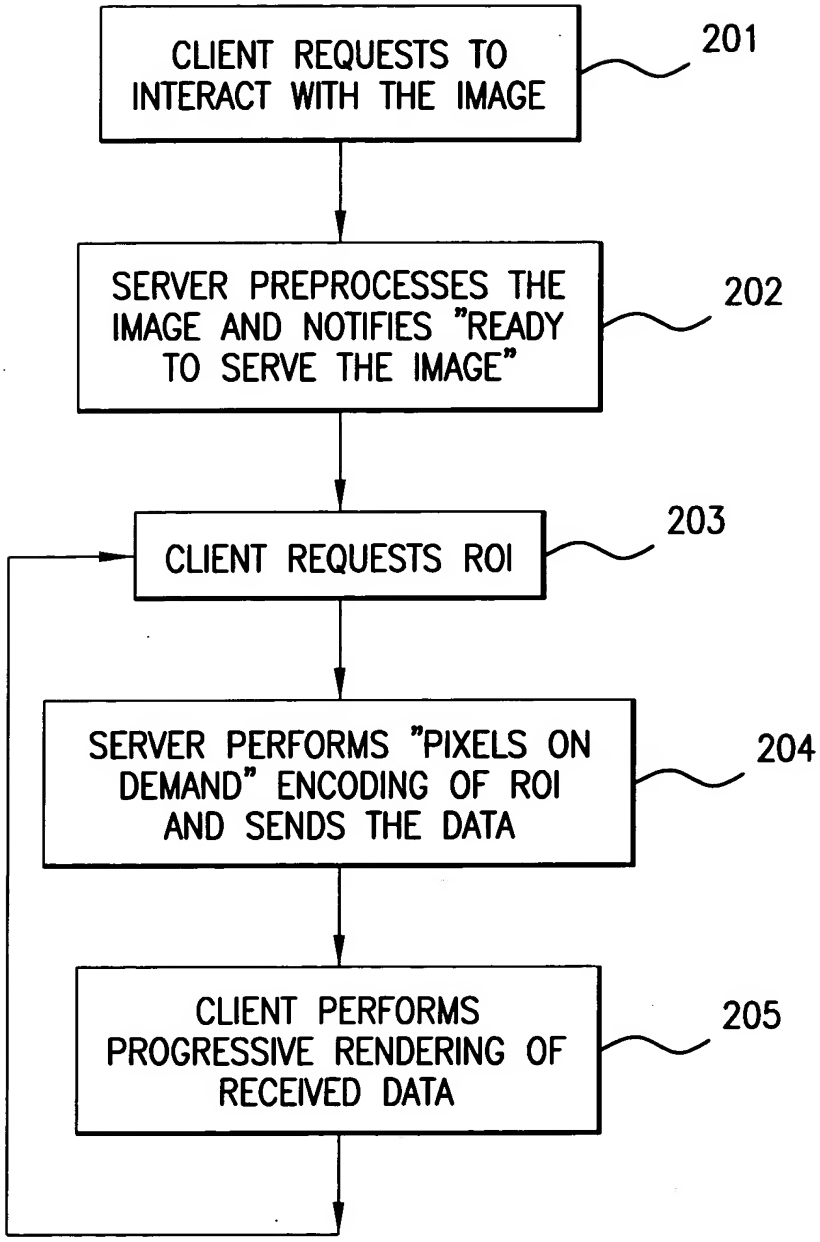

CLIENT REQUESTS TO INTERACT WITH THE IMAGE
201
SERVER PREPROCESSES THE IMAGE AND NOTIFIES "READY TO SERVE THE IMAGE"
202
CLIENT REQUESTS ROI
203
SERVER PERFORMS "PIXELS ON DEMAND" ENCODING OF ROI AND SENDS THE DATA
204
CLIENT PERFORMS PROGRESSIVE RENDERING OF RECEIVED DATA
205

FIG.2

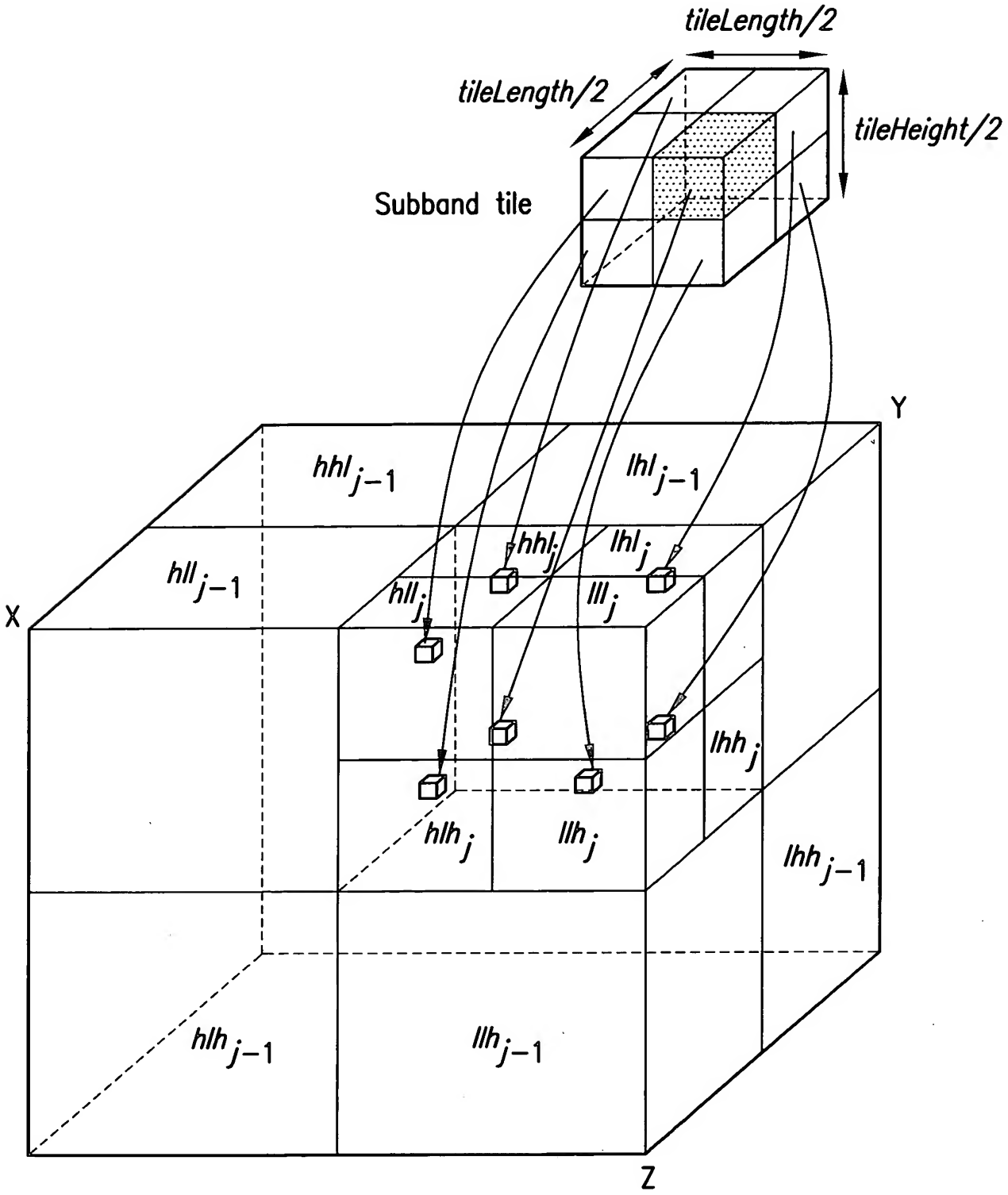

tileLength/2
tileLength/2
tileHeight/2
Subband tile
Y
X
Z
hhl_{j-1}
lhl_{j-1}
hll_{j-1}
hhl_j
lhl_j
hll_j
lll_j
lhh_j
hlh_j
llh_j
lhh_{j-1}
hlh_{j-1}
llh_{j-1}

FIG.3

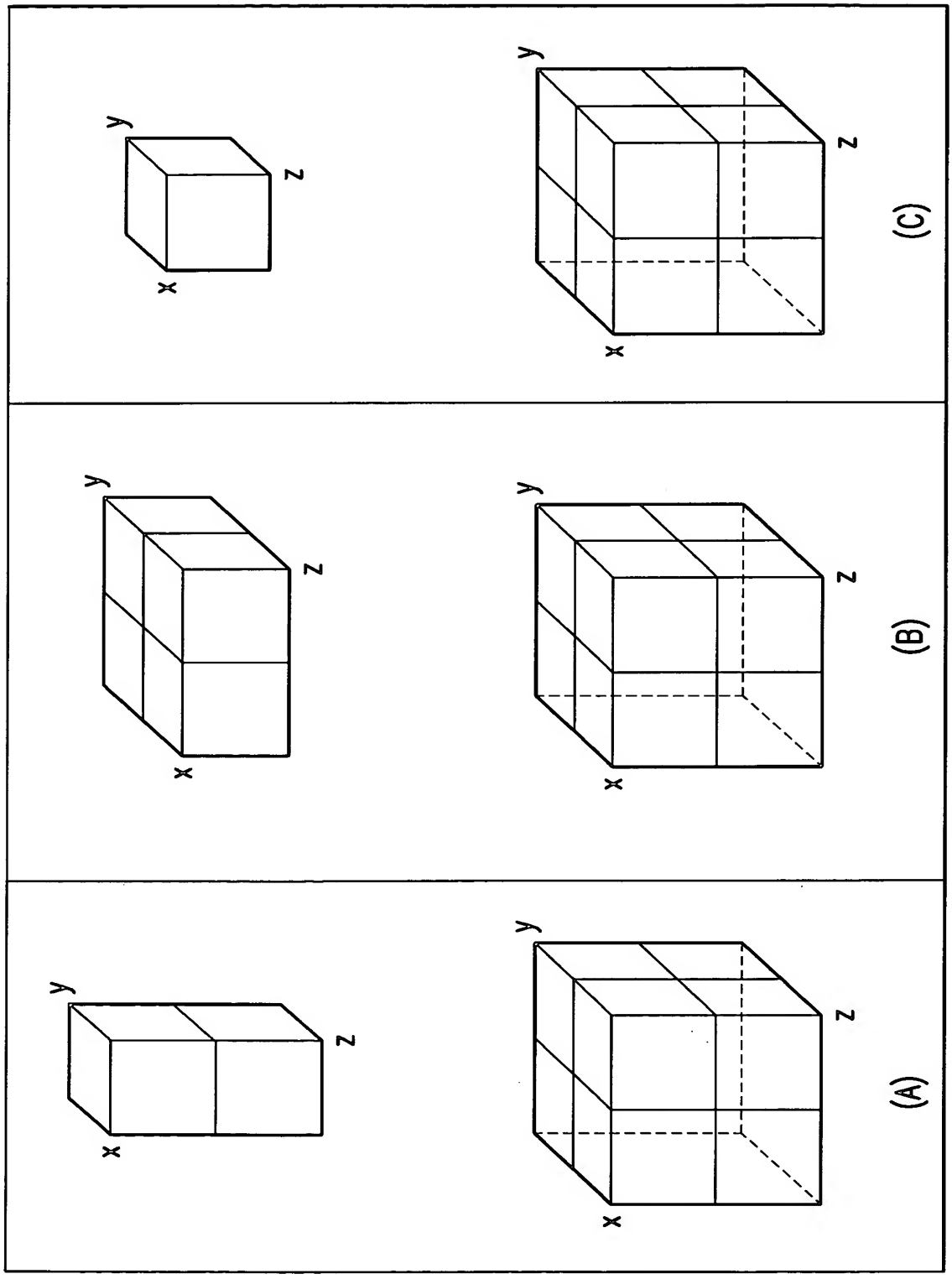


FIG.4

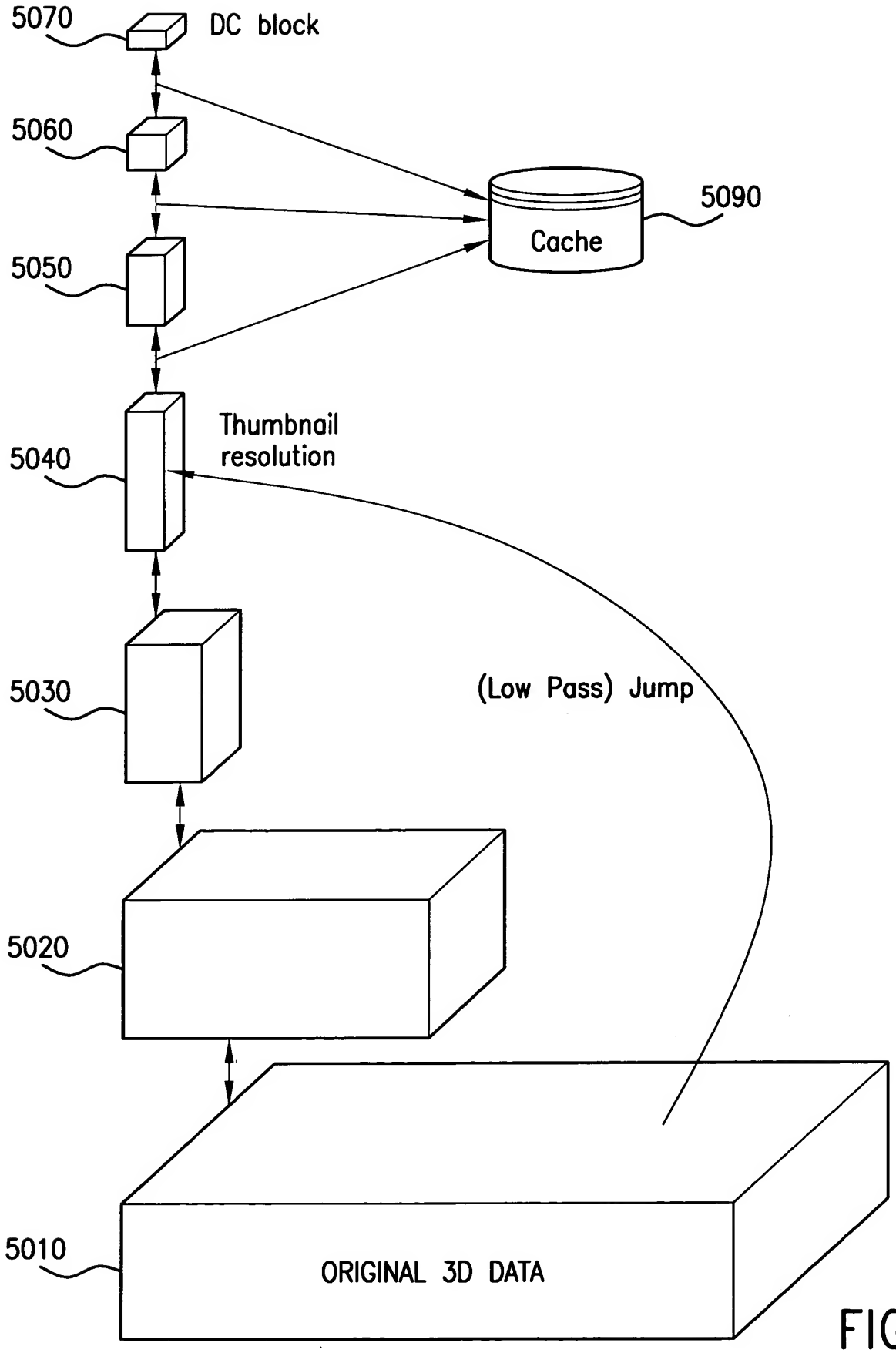

5070
DC block
5060
5090
Cache
5050
Thumbnail
resolution
5040
5030
(Low Pass) Jump
5020
5010
ORIGINAL 3D DATA

FIG.5

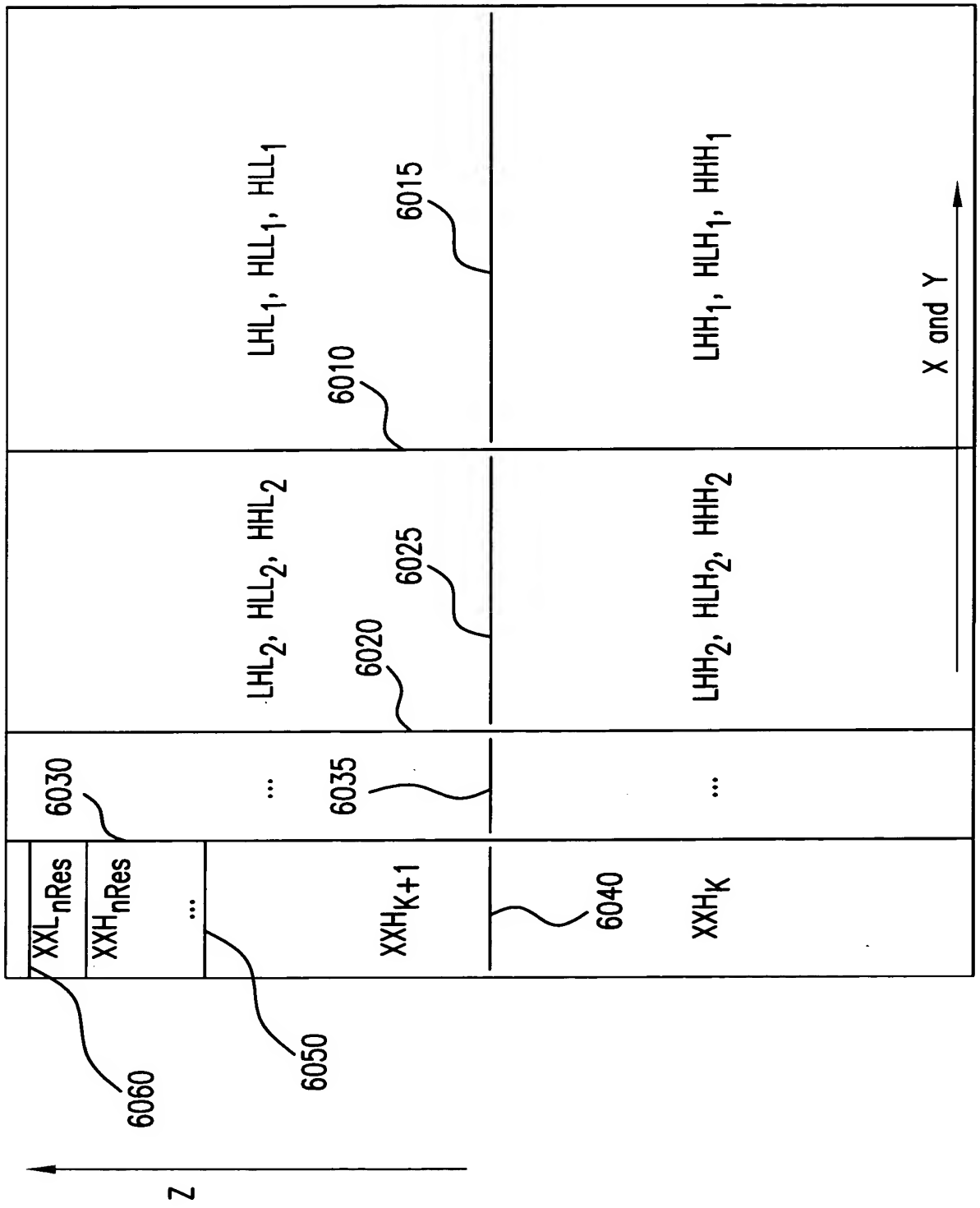
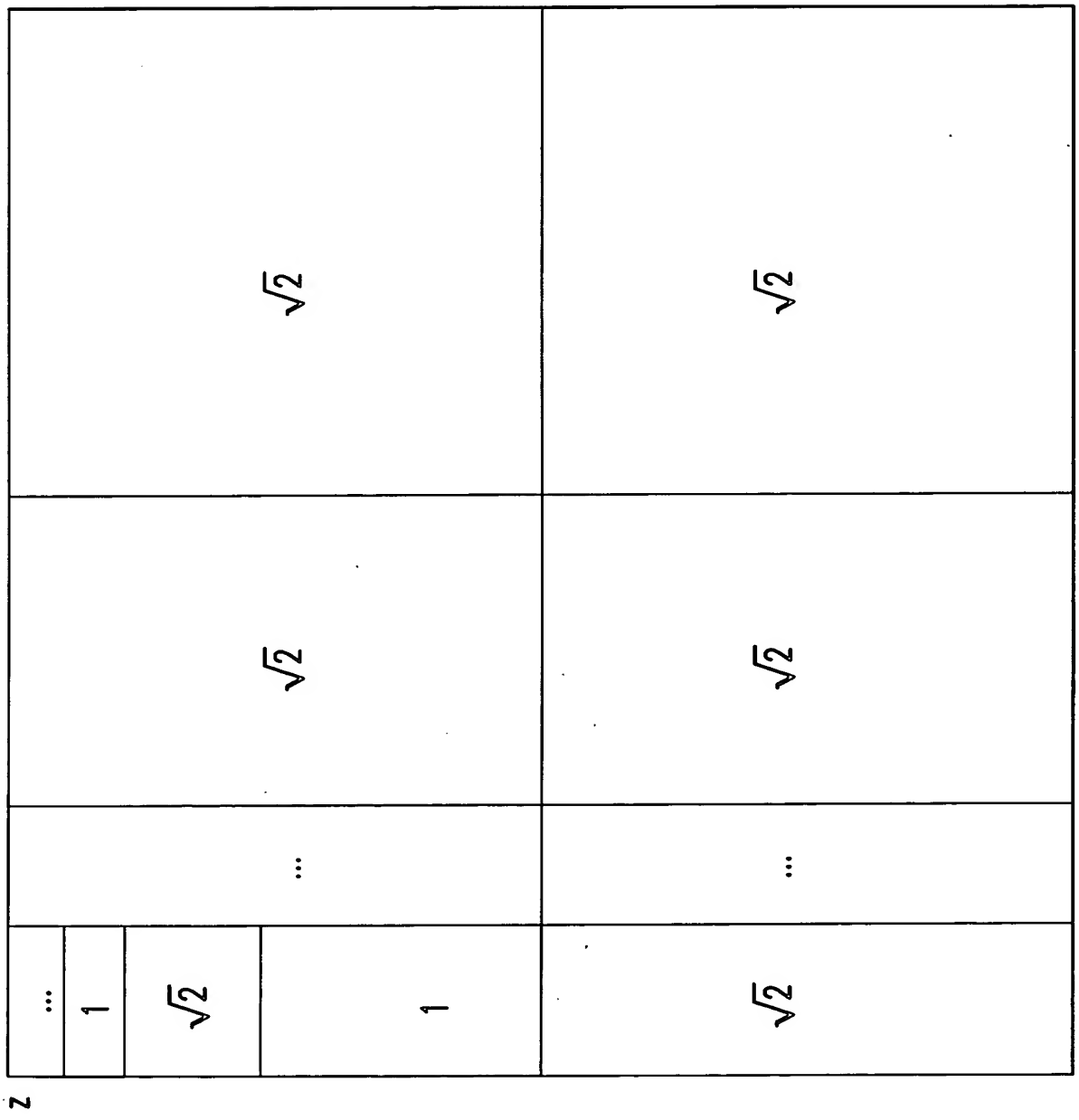

LHL_1, HLL_1, HLL_1
LHH_1, HLH_1, HHH_1
LHL_2, HLL_2, HHL_2
LHH_2, HLH_2, HHH_2
XXH_{K+1}
XXH_K
XXL_{nRes}
XXH_{nRes}
6010
6015
6020
6025
6030
6035
6040
6050
6060
X and Y
Z

FIG.6

FIG.7



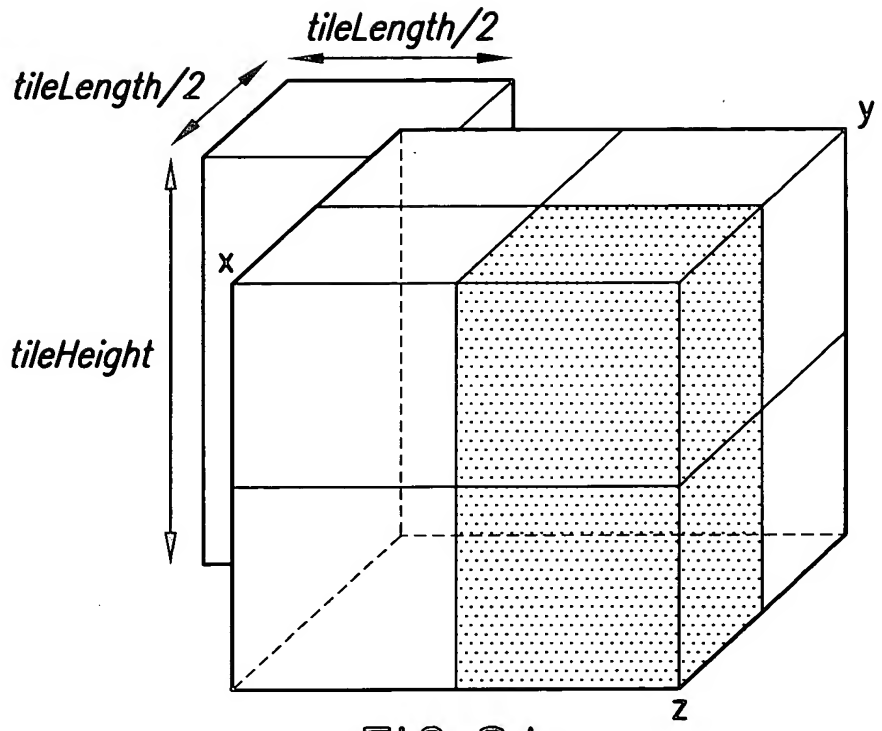

tileLength/2
tileLength/2
y
x
tileHeight
z

FIG.8A

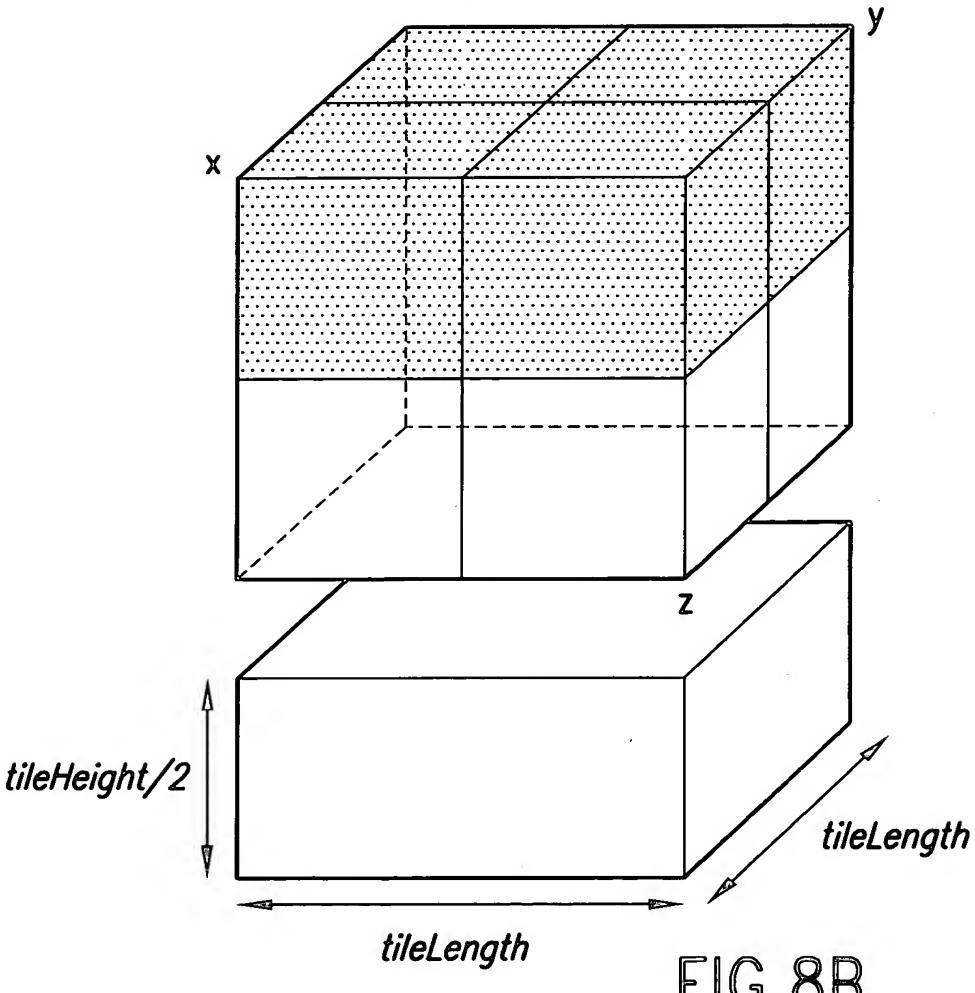

y
x
z
tileHeight/2
tileLength
tileLength

FIG.8B

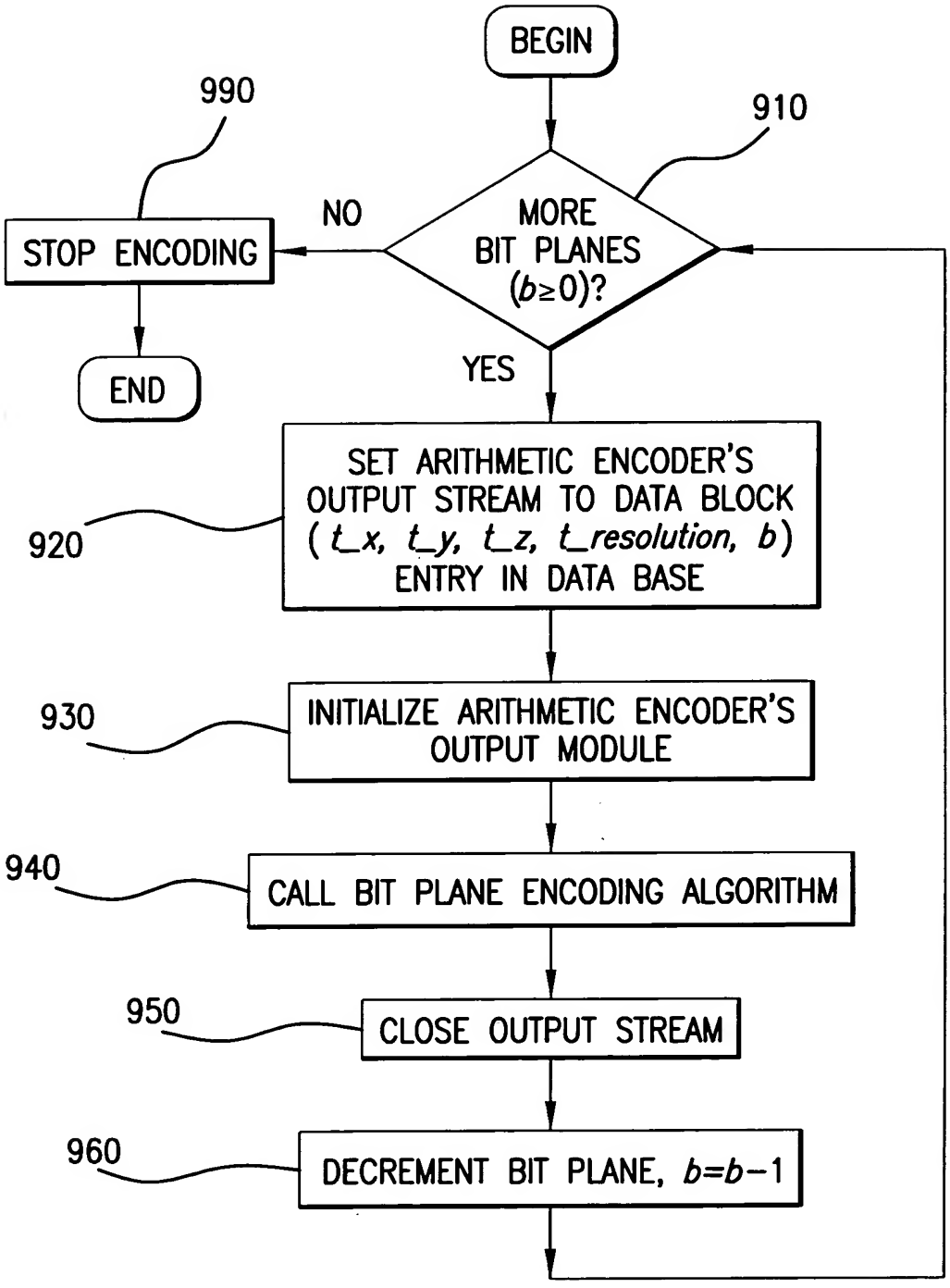

BEGIN
990
910
NO
MORE BIT PLANES ($b \geq 0$)?
STOP ENCODING
END
YES
SET ARITHMETIC ENCODER'S OUTPUT STREAM TO DATA BLOCK (t_x, t_y, t_z, t_resolution, b) ENTRY IN DATA BASE
920
INITIALIZE ARITHMETIC ENCODER'S OUTPUT MODULE
930
CALL BIT PLANE ENCODING ALGORITHM
940
CLOSE OUTPUT STREAM
950
DECREMENT BIT PLANE, $b=b-1$
960

FIG.9

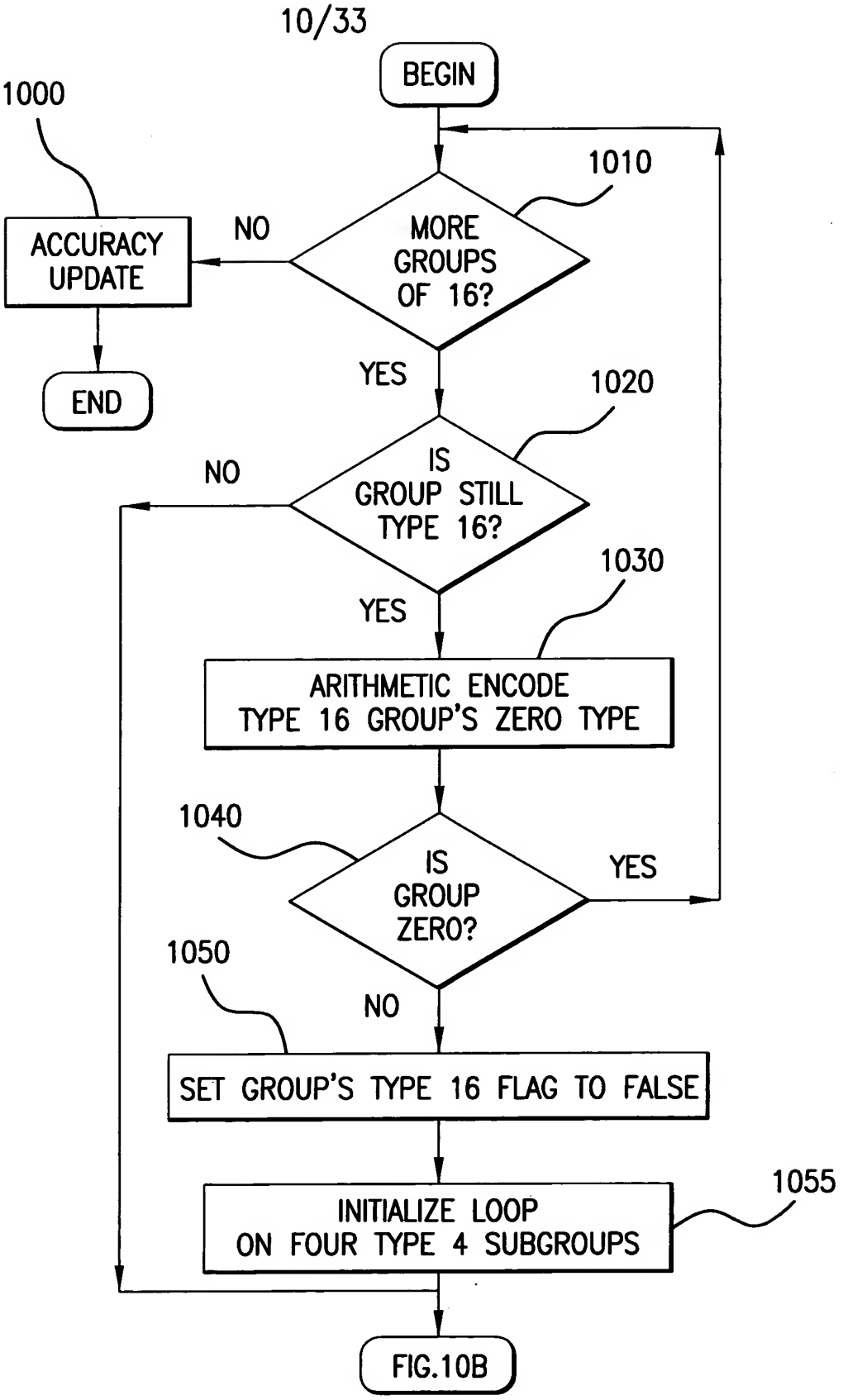

BEGIN
1000
ACCURACY UPDATE
NO
1010
MORE GROUPS OF 16?
YES
END
1020
IS GROUP STILL TYPE 16?
NO
YES
1030
ARITHMETIC ENCODE TYPE 16 GROUP'S ZERO TYPE
1040
IS GROUP ZERO?
YES
1050
NO
SET GROUP'S TYPE 16 FLAG TO FALSE
1055
INITIALIZE LOOP ON FOUR TYPE 4 SUBGROUPS
FIG.10B

FIG.10A

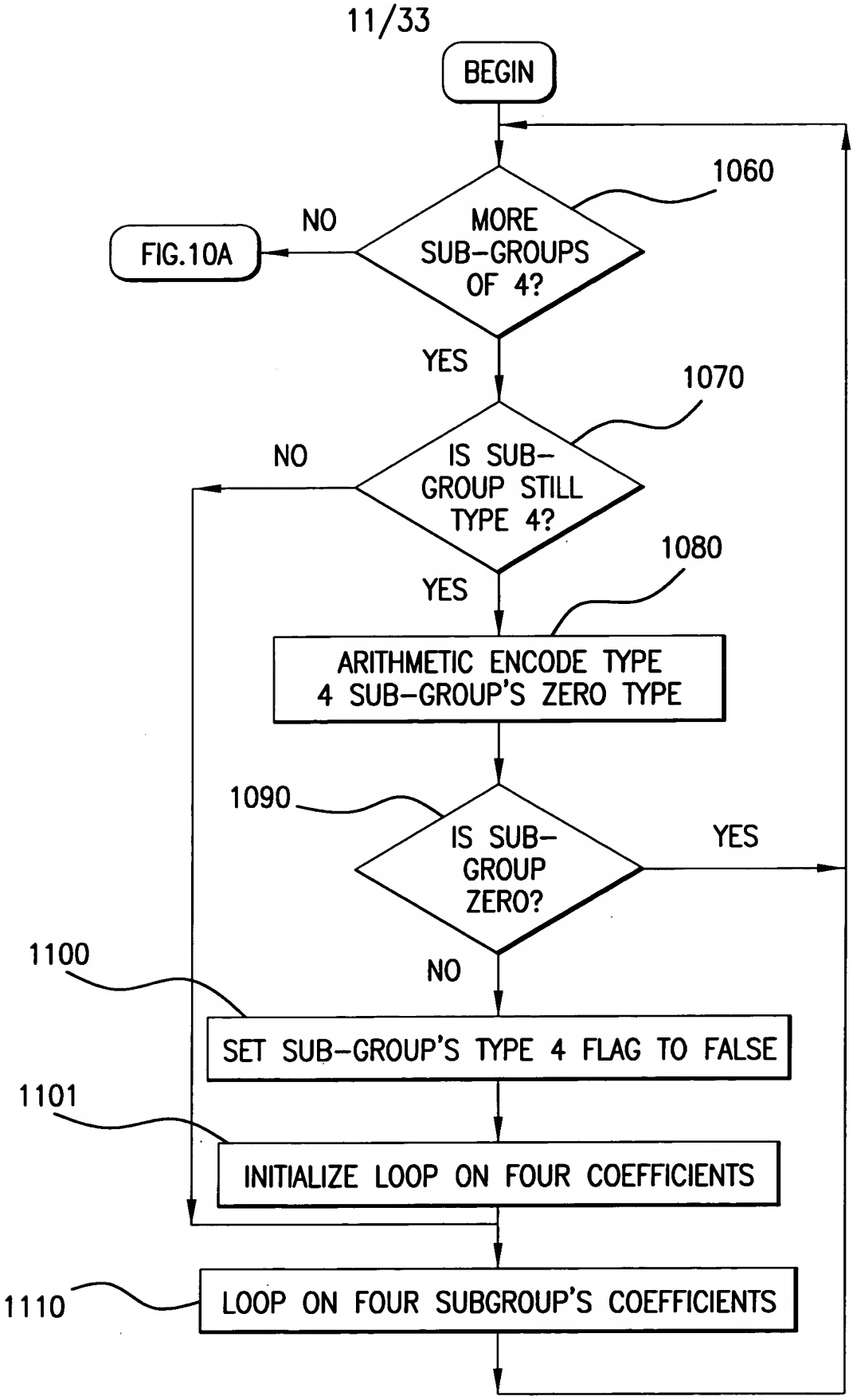

BEGIN
1060
MORE SUB-GROUPS OF 4?
NO
FIG.10A
YES
1070
IS SUB-GROUP STILL TYPE 4?
NO
YES
1080
ARITHMETIC ENCODE TYPE 4 SUB-GROUP'S ZERO TYPE
1090
IS SUB-GROUP ZERO?
YES
NO
1100
SET SUB-GROUP'S TYPE 4 FLAG TO FALSE
1101
INITIALIZE LOOP ON FOUR COEFFICIENTS
1110
LOOP ON FOUR SUBGROUP'S COEFFICIENTS

FIG.10B

```
zeroModel_16.start_model();
zeroModel_4.start_model();
zeroCoefModel.start_model();
coefSignModel.start-model();

while(encoder.getNextGroupOf16()) {
   bool isZero;

   if (encoder.isGroupType16()) {
      isZero = encoder.isZeroGroupOf16();
      arithmetic_encode_symbol(ZeroModel_16,isZero);
      if (isZero)
            continue;
   }

   while (encoder.getNextGroupof4()) {
      if (encoder.isGroupType4()) {
         if (!encoder.mustbeNoZeroGroup()) {
            isZero = encoder.isZeroGroupOf4();
            arithmetic_encode_symbol(ZeroModel_4,isZero);
            if (isZero)
                  continue;
         }
      }

      while (encoder.getNext_Type1_Coef(isZero)) {
         if (!encoder.mustbeNoZeroCoef())
            arithmetic_encode_symbol(zeroCoefModel,isZero);
         if (!isZero)
            arithmetic_encode_symbol(coefSignModel,encoder.getCoefSign));
      }
   }

   if (!(encoder.isLastBitPlane() && equalBinSetting)) {
      bitModel.start_model();

      int bit;
```

FIG. 11

```
bitModel.startModel();
zeroCoefModel.startModel();
coefSignModel.startModel();
while (encoder.moreCoef()) {                                    1210
      if (encoder.isCoefReported()) {                           1220
            arithmetic_encode_symbol(
                  bitModel,encoder.reportedCoefPrecisionBit());
      } else {
            if ( encoder.isCoefExactZero() )                    1230
                  arithmetic_encode_symbol(zeroCoefModel,true);
            else {
                  arithmetic_encode_symbol(zeroCoefModel,false);
                  arithmetic_encode_symbol(
                        coefSignModel,encoder.getCoefSign());
            }
```

FIG. 12A

```
bitModel.startModel();

for (int z = 0 ; z != HalfBitPlaneZSize;z++) {
    for (int y = 0 ; y != HalfBitPlaneYSize;y++) {
        for (int x = 0 ; x != HalfBitPlaneXSize;x++) {
            arithmetic_encode_symbol(bitModel,coefHalfBit[x][y][z]);
        }
    }
}
```

FIG. 12B

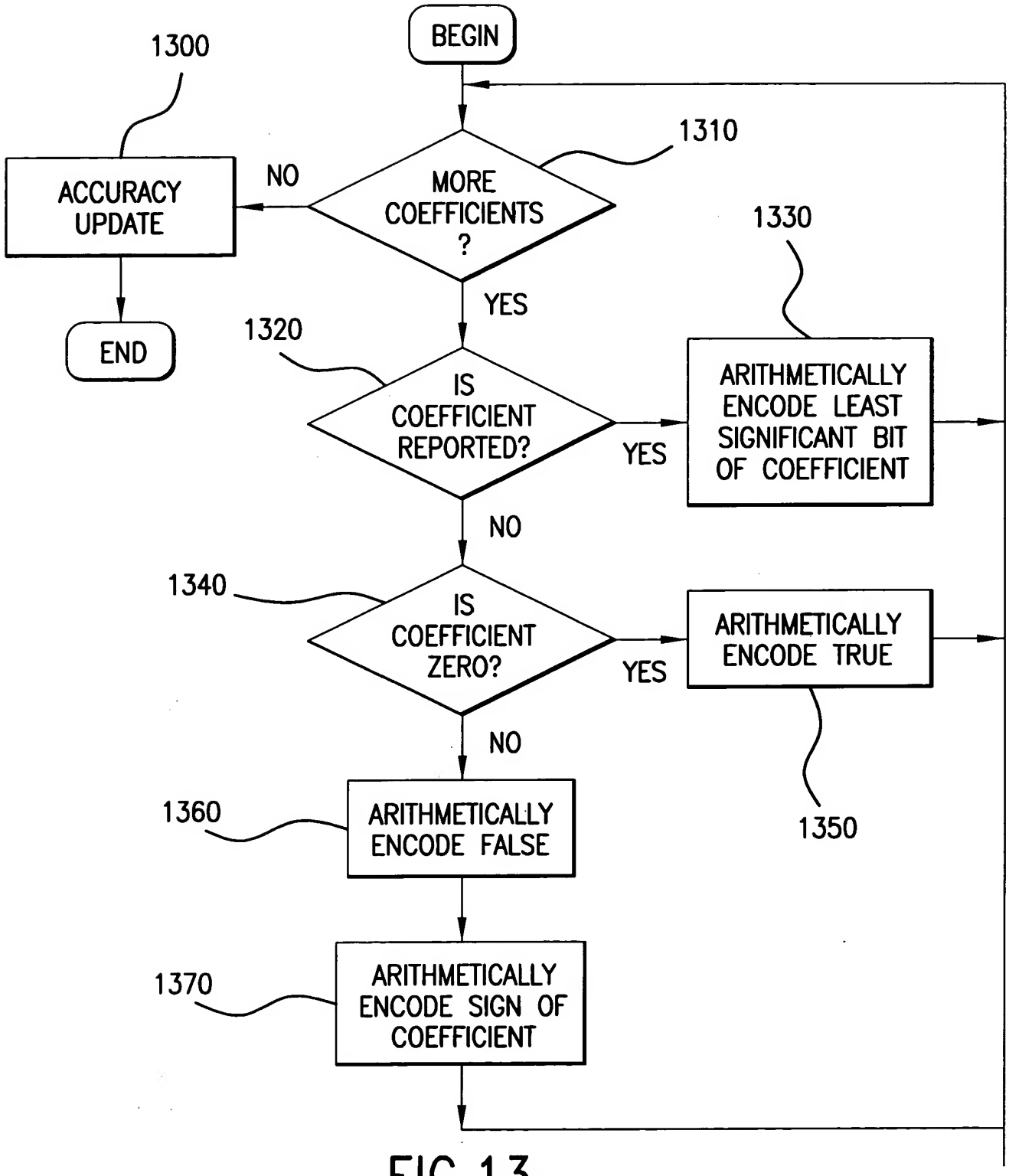

BEGIN
1300
ACCURACY UPDATE
END
NO
MORE COEFFICIENTS ?
1310
YES
1320
IS COEFFICIENT REPORTED?
YES
1330
ARITHMETICALLY ENCODE LEAST SIGNIFICANT BIT OF COEFFICIENT
NO
1340
IS COEFFICIENT ZERO?
YES
ARITHMETICALLY ENCODE TRUE
1350
NO
1360
ARITHMETICALLY ENCODE FALSE
1370
ARITHMETICALLY ENCODE SIGN OF COEFFICIENT

FIG.13

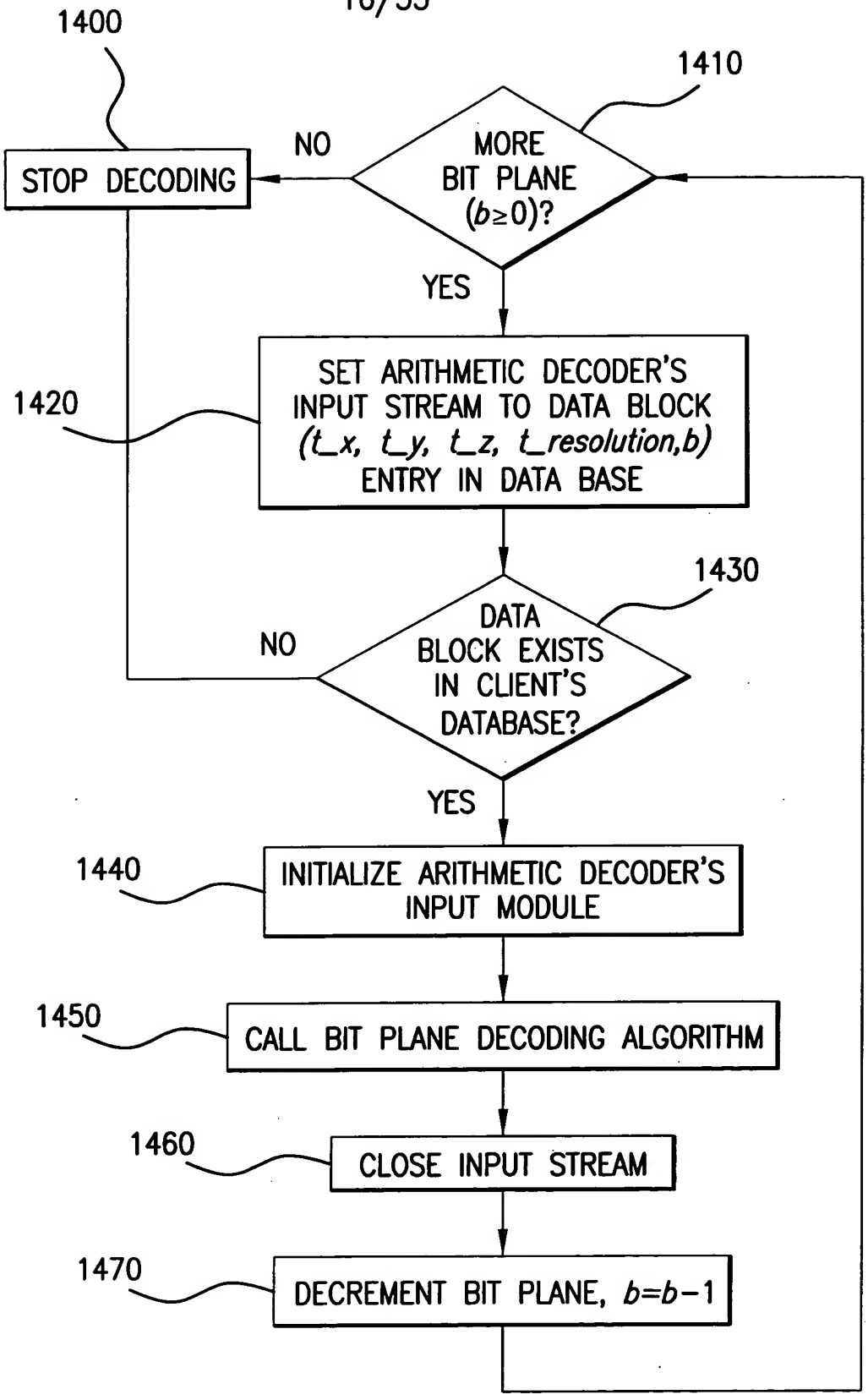

1400
STOP DECODING
NO
1410
MORE BIT PLANE ($b \geq 0$)?
YES
1420
SET ARITHMETIC DECODER'S INPUT STREAM TO DATA BLOCK (t_x, t_y, t_z, t_resolution,b) ENTRY IN DATA BASE
1430
NO
DATA BLOCK EXISTS IN CLIENT'S DATABASE?
YES
1440
INITIALIZE ARITHMETIC DECODER'S INPUT MODULE
1450
CALL BIT PLANE DECODING ALGORITHM
1460
CLOSE INPUT STREAM
1470
DECREMENT BIT PLANE, $b=b-1$

FIG.14

```
zeroModel_16.start_model();
zeroModel_4.start_model();
zeroCoefModel.start_model();
coefSignModel.start_model();

while(decoder.getNextGroupOf16()) {
    if (decoder.isGroupType16()) {
       if (arithmetic_decode_symbol(zeroModel_16)) {
          decoder.zeroGroupOf16();
          continue;
       }
       else
           decoder.removeZeroGroupOf16();
    }

    while (decoder.getNextGroupOf4()) {
      if (decoder.isGroupType4()) {
         if (!decoder.mustbeNotZeroGroup()) {
               if (arithmetic_decode_symbol(zeroModel_4)) {
                  decoder.zeroGroupOf4();
                  continue;
               }
         }
         decoder.removeZeroGroupOf4();
      }

      while (decoder.getNext_Type1_Coef()) {
        if (decoder.mustbeNotZeroCoef())

decoder.setNextSigCoef(arithmetic_decode_symbol(coefSignmodel));
        else if (!arithmetic_decode_symbol(zeroCoefModel))

      decoder.setNextSigCoef(arithmetic_decode_symbol(coefSignmodel));
      }
   }
}

if (! (decoder.isLastBitPlane() && equalBinSetting)) {
   bitModel.start_model();
   while(decoder.moreSignificantCoef())
      decoder.setSignificantCoefBit(arithmetic_decode_symbol(bitModel));
}
```

FIG. 15

```
bitModel .startModel();
zeroCoefModel.startmodel();
coefSignModel.startmodel();

decoder.initializeLSBPlaneCoefScan();                    ~1610

while (decoder.moreCoef()) {
      if (decoder.isCoefReported()) {
            if(decoder.isSkippedCoef()) {
                  decoder. updateLSB (0);
            }
            else {

decoder.updateLSB(arithmetic_decoder_symbol(bitModel));
            }
      }
      else {
             if(!decoder.isSkippedCoef()) {
                   if (!arithmetic_decoder_symbol(zeroCoefModel))

decoder.setLSB(arithmetic_decoder_symbol(coefSignModel));
             }
```

FIG. 16A

```
bitModel.startModel();

for (int z = 0 ; z != HalfBitPlaneZSize;z++) {
    for (int y = 0 ; y != HalfBitPlaneYSize;y++) {
        for (int x = 0 ; x != HalfBitPlaneXSize;x++) {
            coefHalfBit[x][y][z] = arithmetic_decoder_symbol(bitModel);
        }
    }
}
```

FIG. 16B

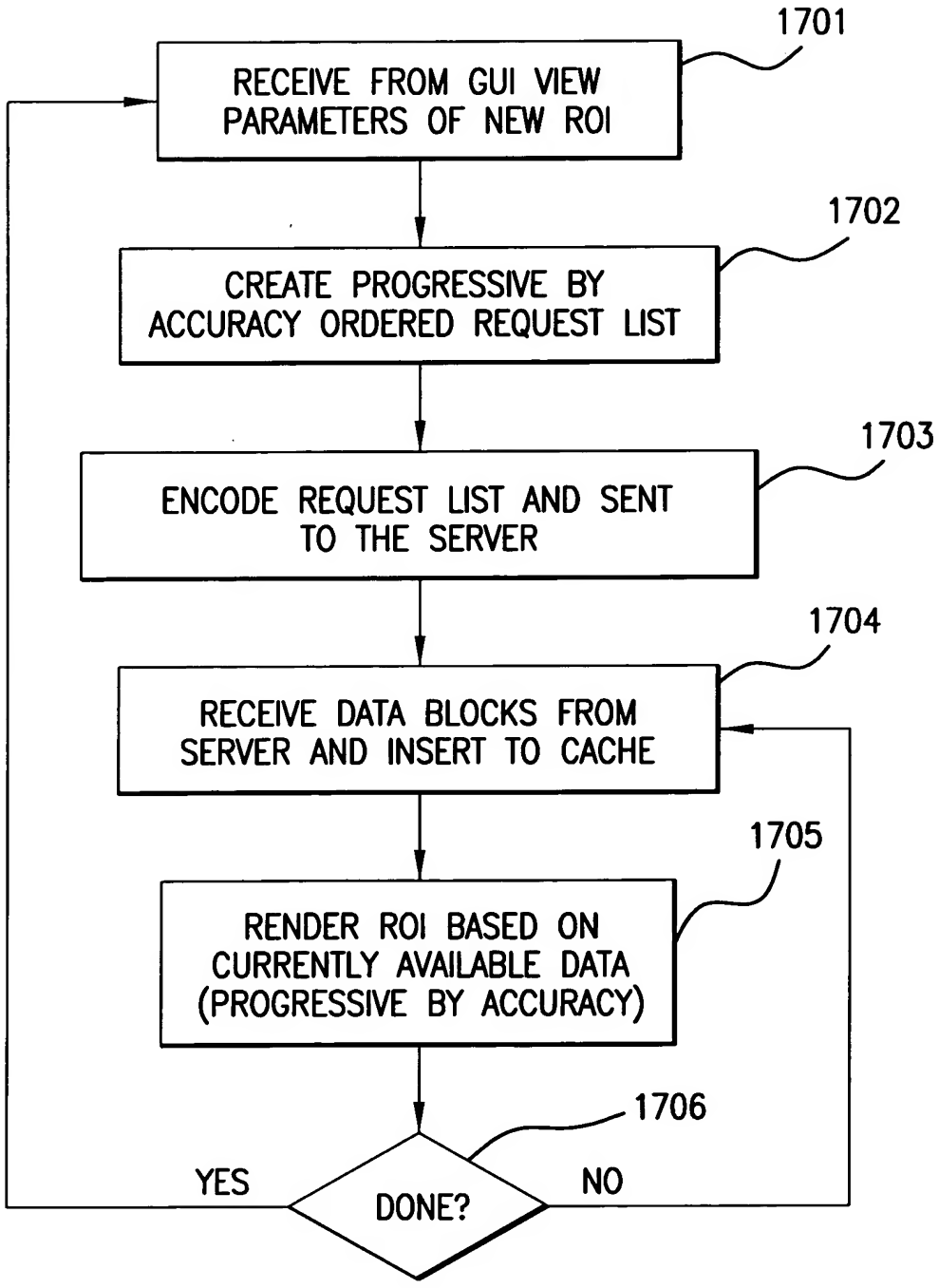

1701
RECEIVE FROM GUI VIEW PARAMETERS OF NEW ROI
1702
CREATE PROGRESSIVE BY ACCURACY ORDERED REQUEST LIST
1703
ENCODE REQUEST LIST AND SENT TO THE SERVER
1704
RECEIVE DATA BLOCKS FROM SERVER AND INSERT TO CACHE
1705
RENDER ROI BASED ON CURRENTLY AVAILABLE DATA (PROGRESSIVE BY ACCURACY)
1706
DONE?
YES
NO

FIG.17

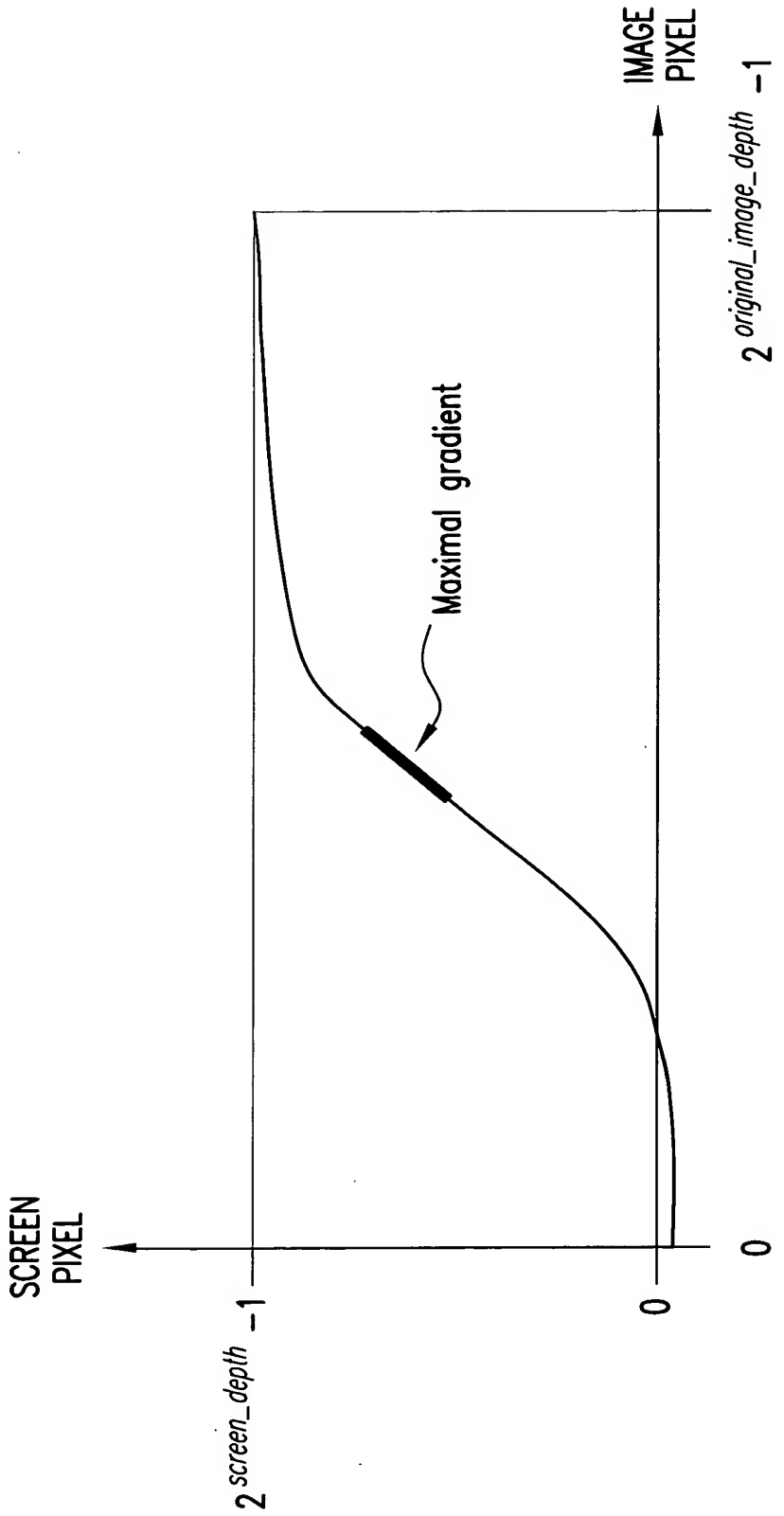

SCREEN PIXEL
IMAGE PIXEL
$2^{screen\_depth} - 1$
0
0
$2^{original\_image\_depth} - 1$
Maximal gradient

FIG.18

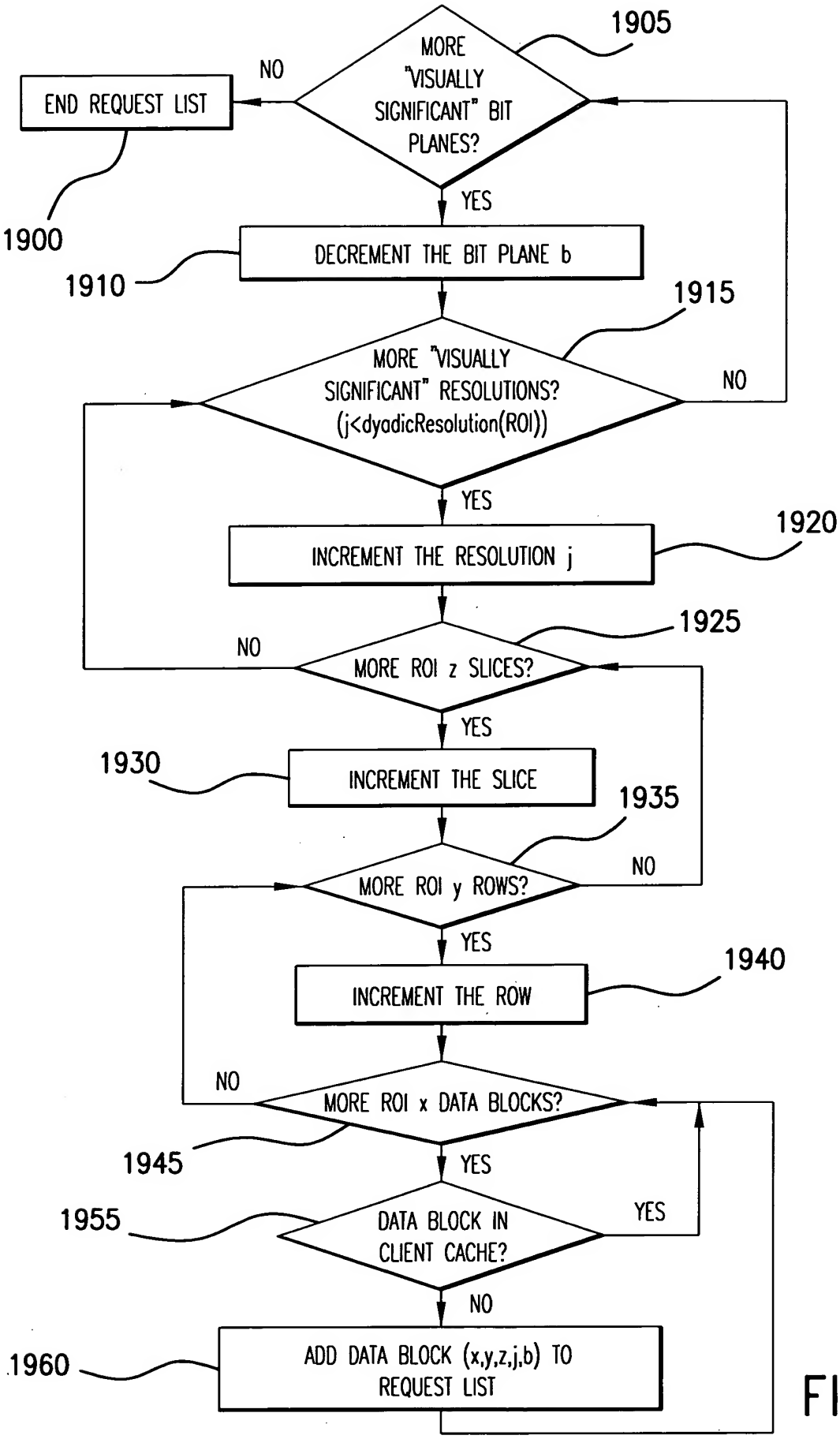

1905
MORE "VISUALLY SIGNIFICANT" BIT PLANES?
NO
END REQUEST LIST
1900
YES
DECREMENT THE BIT PLANE b
1910
1915
MORE "VISUALLY SIGNIFICANT" RESOLUTIONS? (j<dyadicResolution(ROI))
NO
YES
INCREMENT THE RESOLUTION j
1920
1925
NO
MORE ROI z SLICES?
YES
1930
INCREMENT THE SLICE
1935
MORE ROI y ROWS?
NO
YES
INCREMENT THE ROW
1940
NO
MORE ROI x DATA BLOCKS?
1945
YES
1955
DATA BLOCK IN CLIENT CACHE?
YES
NO
1960
ADD DATA BLOCK (x,y,z,j,b) TO REQUEST LIST

FIG.19

```
for (int res = 1 ; res <= dyadicResolution(ROI); res++) {

      for( int z=0;
           z < NumberOfZtilesOnDyadicResolution (res,ROI);
           z++ ) {

           GetCoefficientsofLowerResolution(res, Ztile);

           for( int x=0;
                x < NumberOfXtilesOnDyadicResolution(res,ROI);
                x++ ) {
                for( int y=0;
                     y <
                     NumberOfYtilesOnDyadicResolution(res,ROI);
                     y++ ) {
                     DecodeOrExtractFromCacheSubbandCoefficients
                     ( res, x, y, z );
                }
            }

            ExecuteInverseSubbandTransform(z);

            if( res == dyadicResolution(ROI))
                  ImageResizeAndMappingTo8bitScreen();
      }
```

FIG. 20

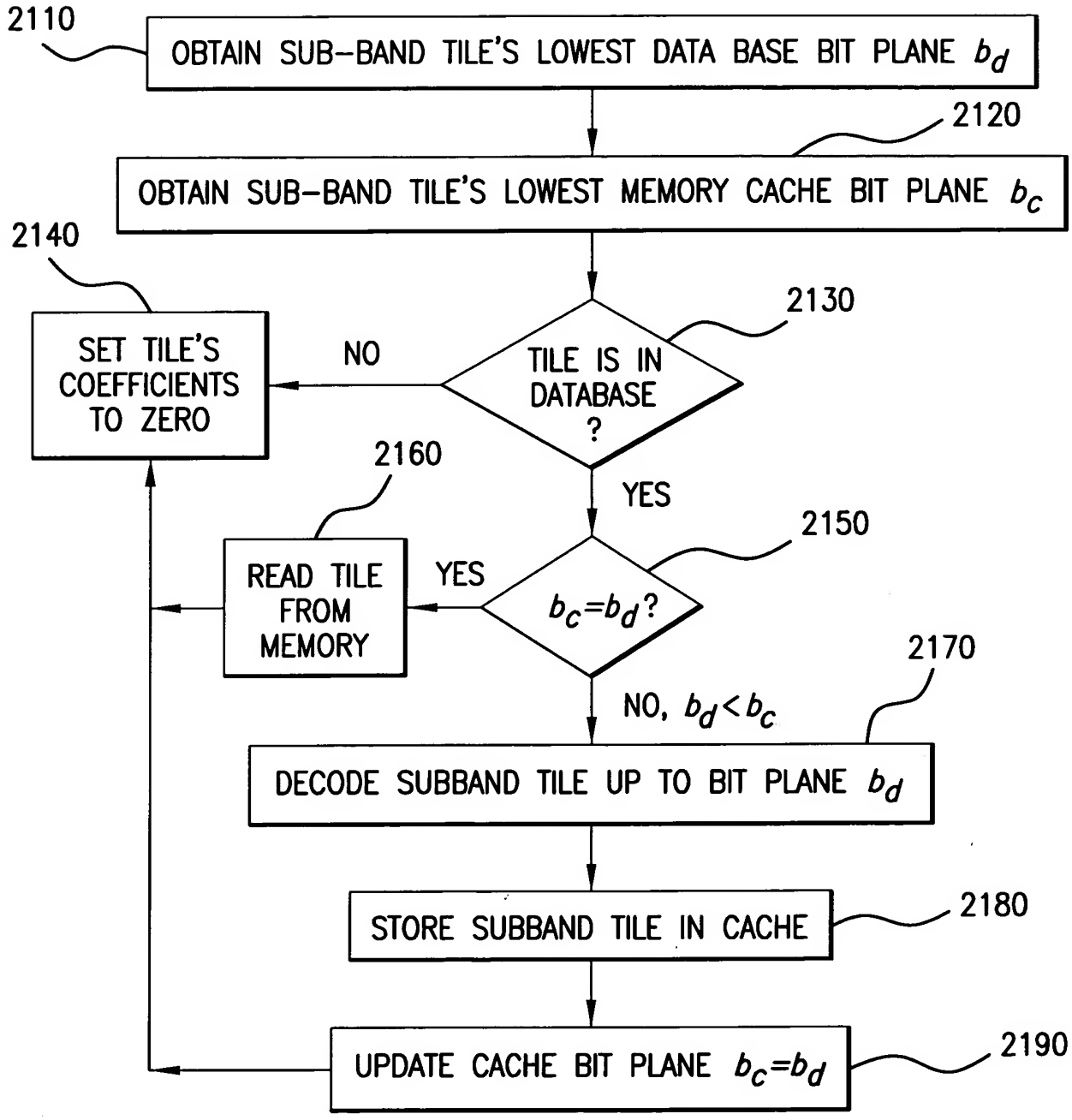

2110
OBTAIN SUB-BAND TILE'S LOWEST DATA BASE BIT PLANE b_d
2120
OBTAIN SUB-BAND TILE'S LOWEST MEMORY CACHE BIT PLANE b_c
2140
SET TILE'S COEFFICIENTS TO ZERO
NO
2130
TILE IS IN DATABASE ?
YES
2160
READ TILE FROM MEMORY
YES
2150
$b_c=b_d$?
NO, $b_d<b_c$
2170
DECODE SUBBAND TILE UP TO BIT PLANE b_d
STORE SUBBAND TILE IN CACHE
2180
UPDATE CACHE BIT PLANE $b_c=b_d$
2190

FIG.21

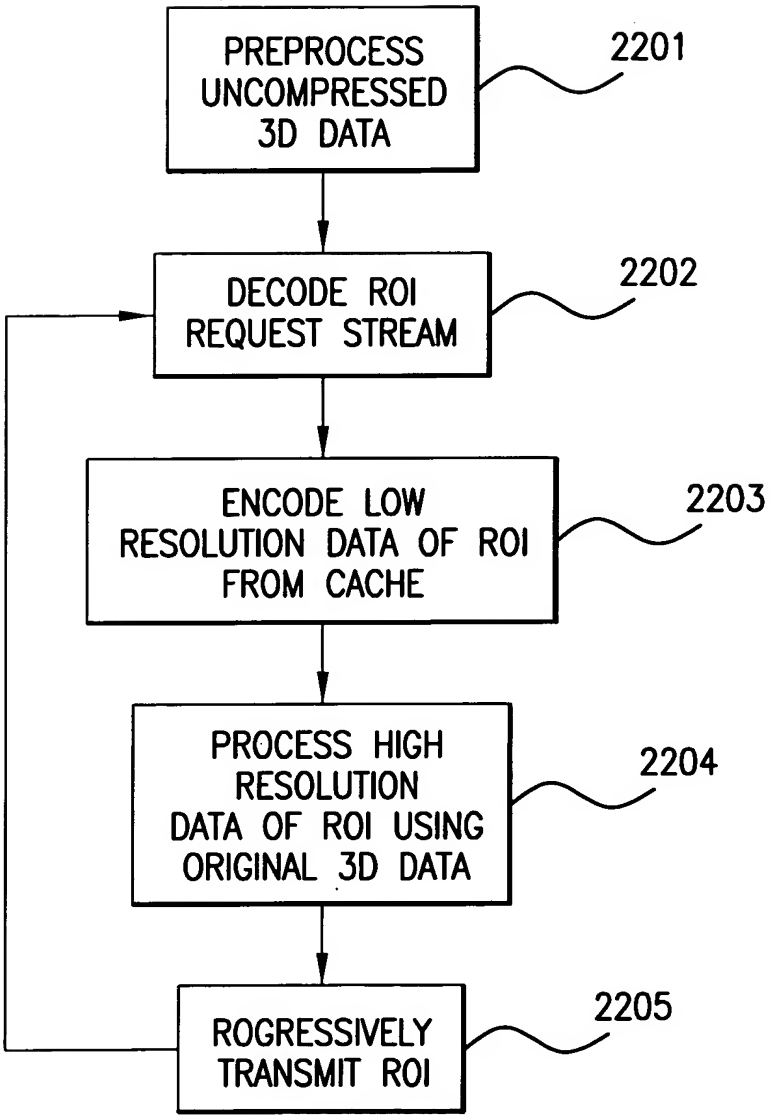

PREPROCESS UNCOMPRESSED 3D DATA
2201
DECODE ROI REQUEST STREAM
2202
ENCODE LOW RESOLUTION DATA OF ROI FROM CACHE
2203
PROCESS HIGH RESOLUTION DATA OF ROI USING ORIGINAL 3D DATA
2204
ROGRESSIVELY TRANSMIT ROI
2205

FIG.22

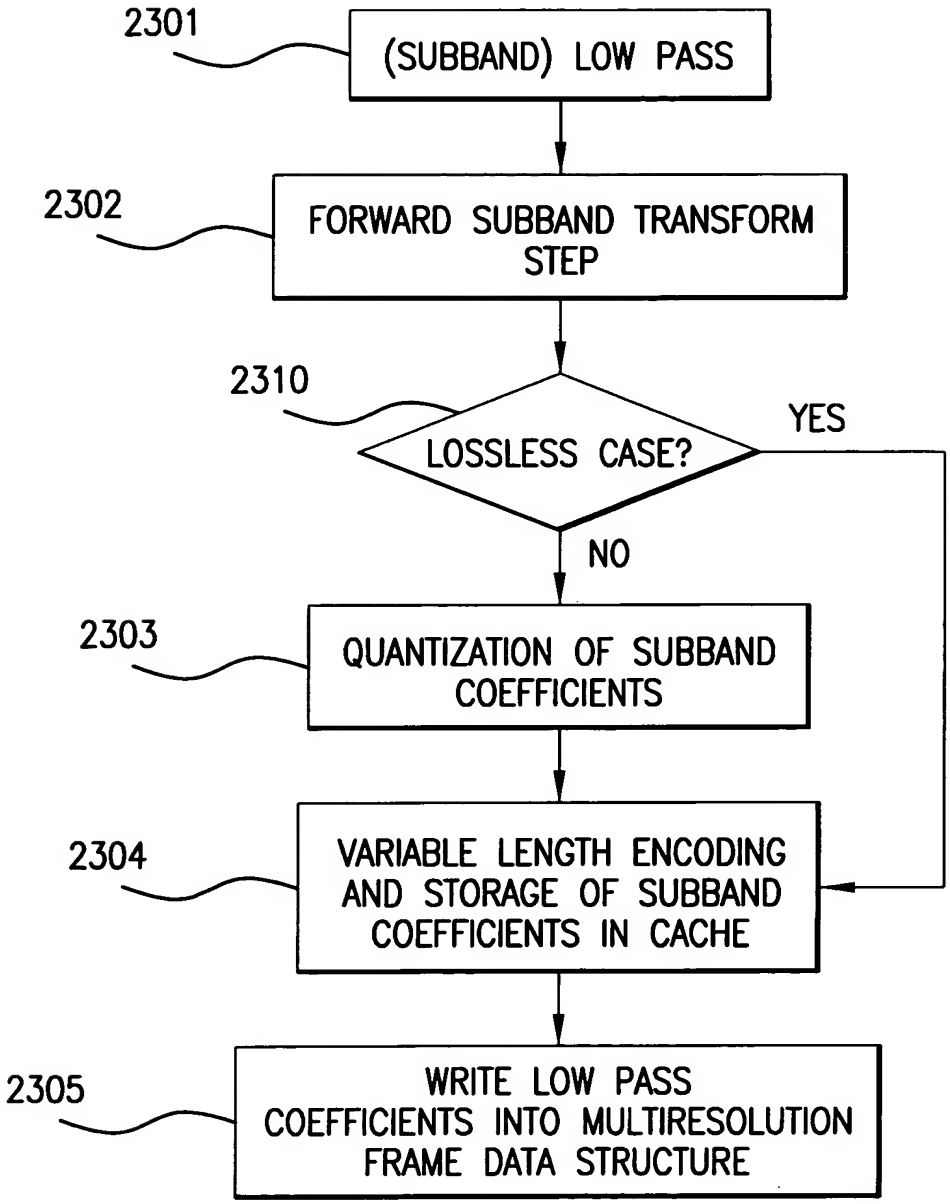

2301
(SUBBAND) LOW PASS
2302
FORWARD SUBBAND TRANSFORM STEP
2310
LOSSLESS CASE?
YES
NO
2303
QUANTIZATION OF SUBBAND COEFFICIENTS
2304
VARIABLE LENGTH ENCODING AND STORAGE OF SUBBAND COEFFICIENTS IN CACHE
2305
WRITE LOW PASS COEFFICIENTS INTO MULTIRESOLUTION FRAME DATA STRUCTURE

FIG.23

```
for(int t_Resolution=numberOfResolutions-jumpSize; t_Resolution>=1 ;
t_Resolution--) {
      leftTilesZInMemoryBuffer(t_Resolution)=
                  NumberOfTilesZInFrameMemoryBuffer(t_Resolution);
      currentTile(t_Resolution)=0;
}

for(t_Resolution=numberOfResolutions-jumpSize; ;) {
      // calculate the Z and it's resolution
      if (currentTile(t_Resolution)< nTileZ(t_Resolution)) {
          for (int t_y = 0 ; t_y < nTileY(t_Resolution); t_y++)
          for (int t_x = 0 ; t_x < nTileX(t_Resolution); t_x++)
               preprocessSubbandTile(t_x,t_y,
currentTile(t_Resolution), t_Resolution);
      }

      // update the indeces
      leftTilesZInMemoryBuffer(t_Resolution)--;
      currentTile(t_Resolution)++;

      if(currentTile(t_Resolution) < nTileZ(t_Resolution)) {
            // switch the resolution
            if(leftTilesZInMemoryBuffer(t_Resolution)==0) {
                  leftTilesZInMemoryBuffer(t_Resolution) =
                        NumberOfTilesZInFrameMemoryBuffer(t_Resolution
                  );
                  t_Resolution --;
            }
            else
                  t_Resolution = numberOfResolutions-jumpSize;
      }
      else }
            t_Resolution --;
            if (t_Resolution<1)
```

FIG. 24

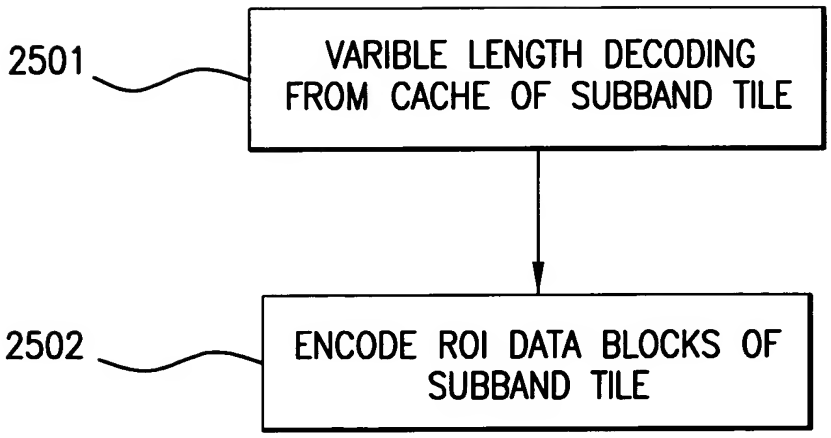

2501
VARIBLE LENGTH DECODING FROM CACHE OF SUBBAND TILE
2502
ENCODE ROI DATA BLOCKS OF SUBBAND TILE

FIG.25

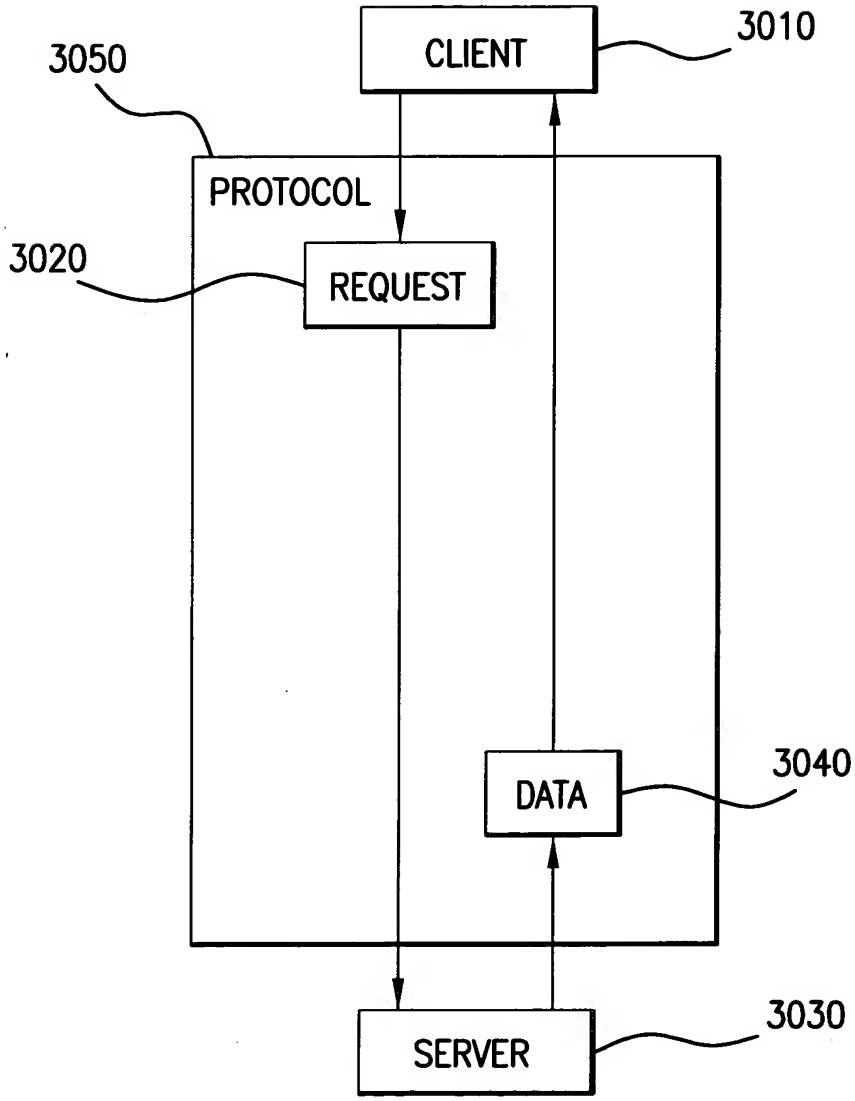

CLIENT
3010
3050
PROTOCOL
3020
REQUEST
DATA
3040
SERVER
3030

FIG.26

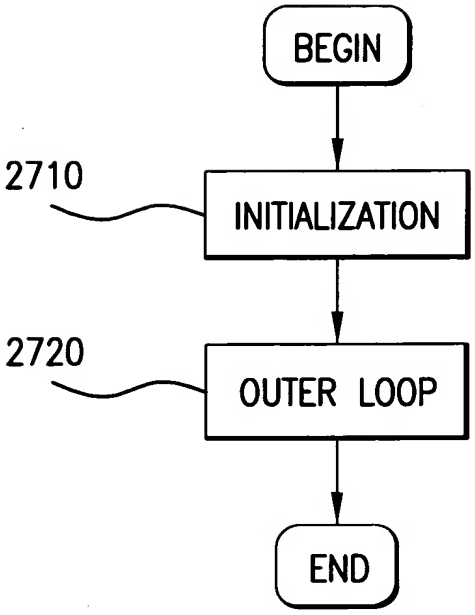


FIG.27

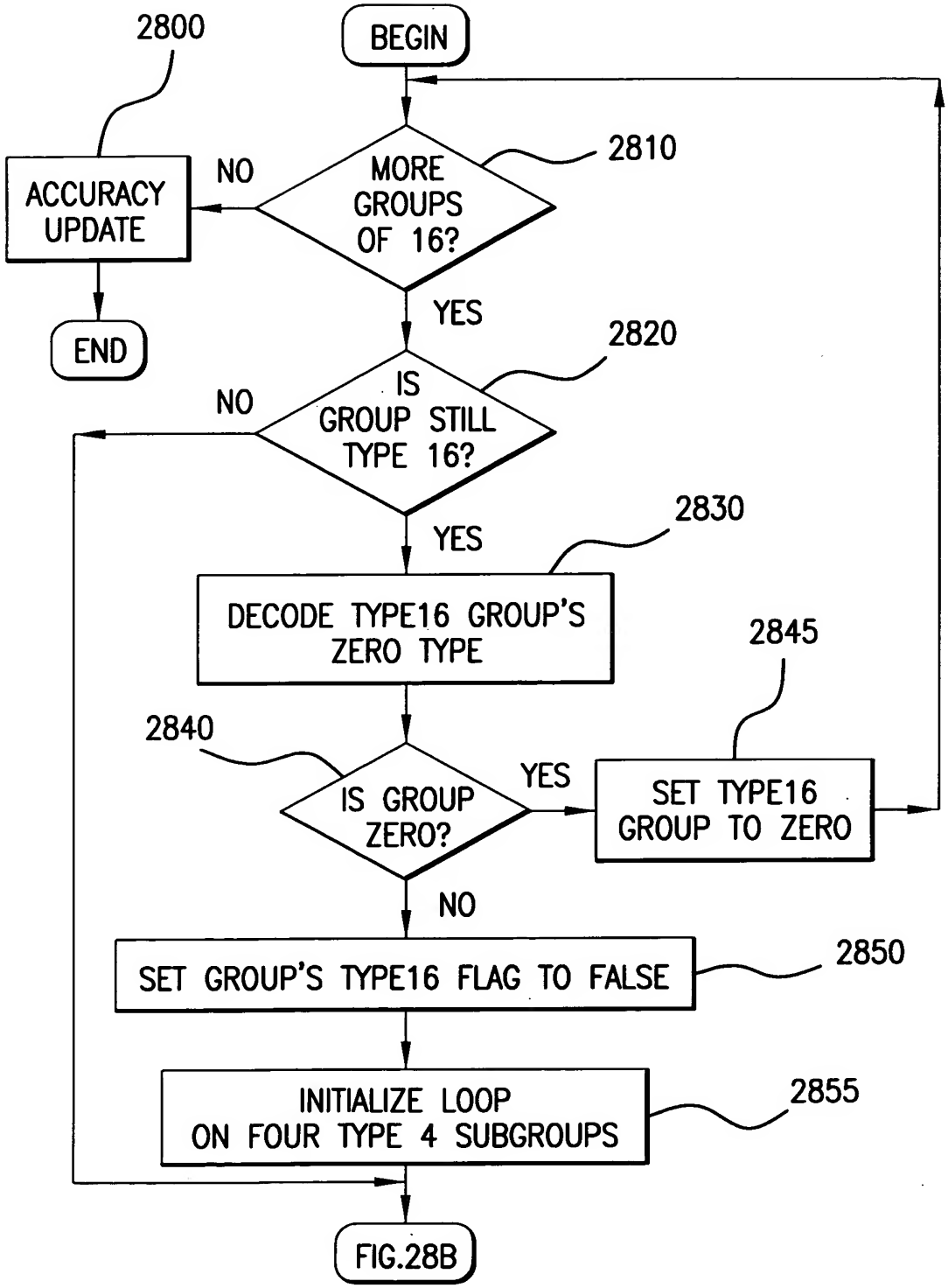

BEGIN
2800
ACCURACY UPDATE
NO
MORE GROUPS OF 16?
2810
YES
END
2820
IS GROUP STILL TYPE 16?
NO
YES
2830
DECODE TYPE16 GROUP'S ZERO TYPE
2845
2840
IS GROUP ZERO?
YES
SET TYPE16 GROUP TO ZERO
NO
SET GROUP'S TYPE16 FLAG TO FALSE
2850
INITIALIZE LOOP ON FOUR TYPE 4 SUBGROUPS
2855
FIG.28B

FIG.28A

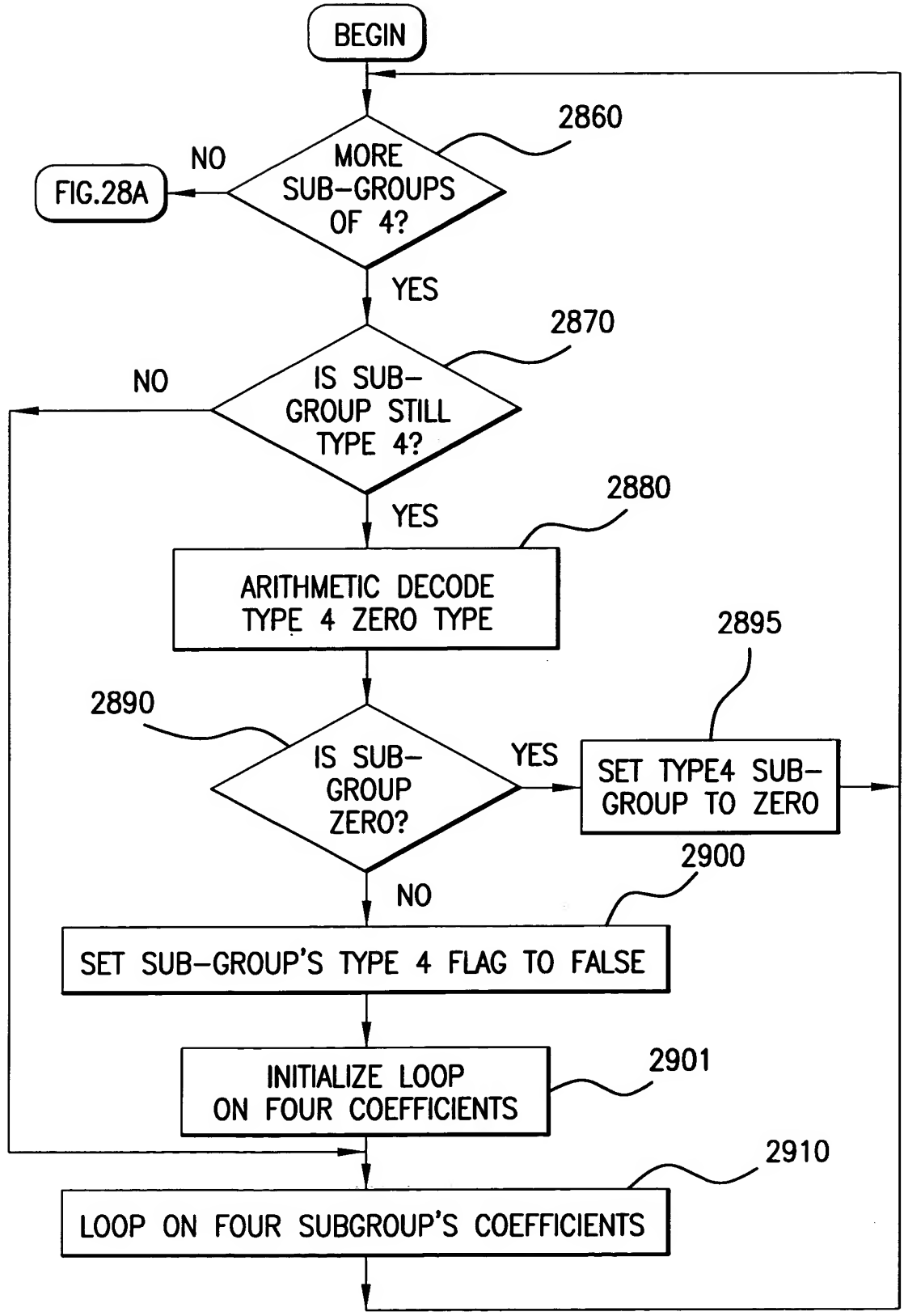

BEGIN
2860
MORE SUB-GROUPS OF 4?
NO
FIG.28A
YES
2870
IS SUB-GROUP STILL TYPE 4?
NO
YES
2880
ARITHMETIC DECODE TYPE 4 ZERO TYPE
2895
2890
IS SUB-GROUP ZERO?
YES
SET TYPE4 SUB-GROUP TO ZERO
2900
NO
SET SUB-GROUP'S TYPE 4 FLAG TO FALSE
INITIALIZE LOOP ON FOUR COEFFICIENTS
2901
2910
LOOP ON FOUR SUBGROUP'S COEFFICIENTS

FIG.28B

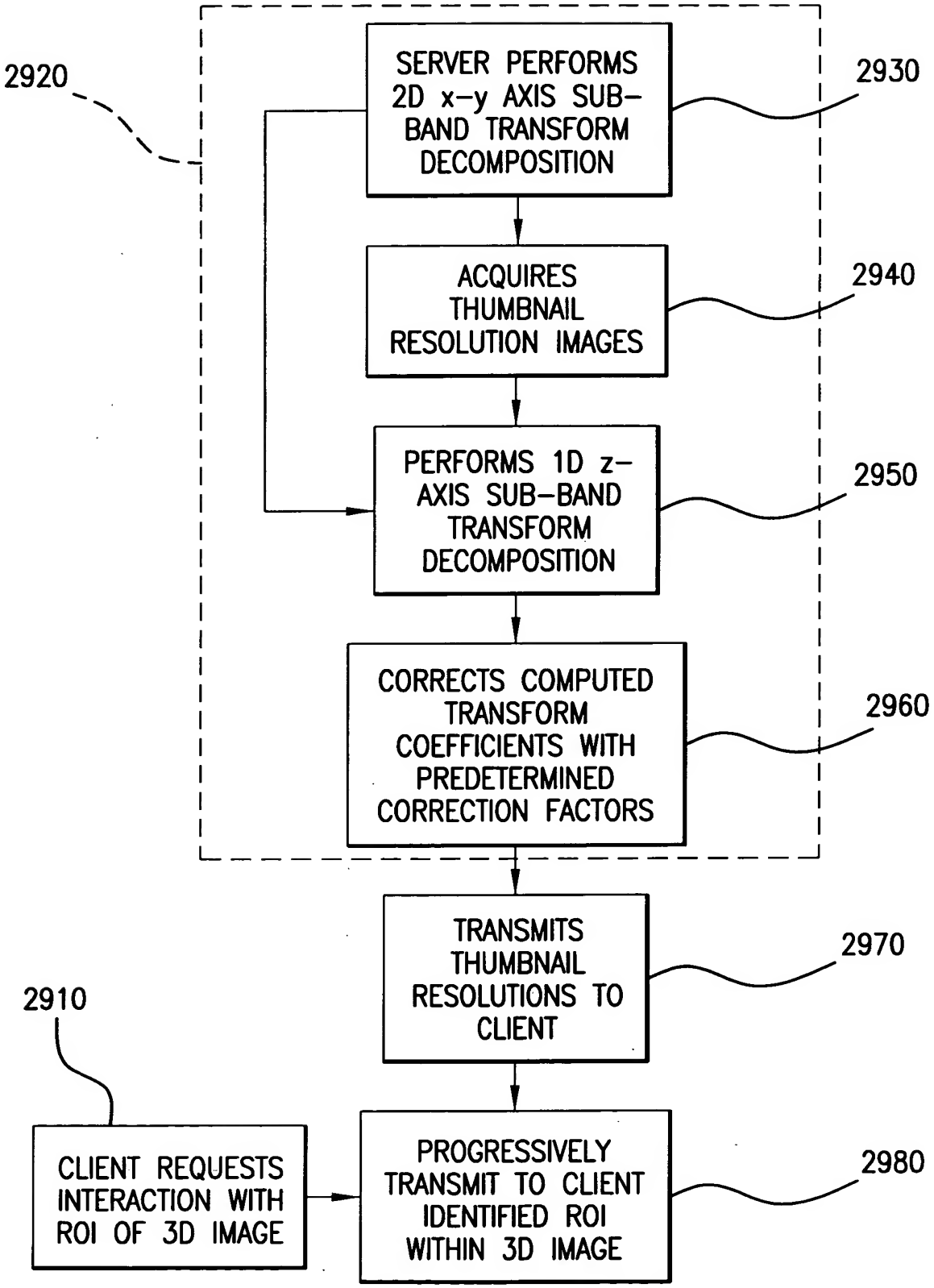

2920
SERVER PERFORMS 2D x-y AXIS SUB-BAND TRANSFORM DECOMPOSITION
2930
ACQUIRES THUMBNAIL RESOLUTION IMAGES
2940
PERFORMS 1D z-AXIS SUB-BAND TRANSFORM DECOMPOSITION
2950
CORRECTS COMPUTED TRANSFORM COEFFICIENTS WITH PREDETERMINED CORRECTION FACTORS
2960
TRANSMITS THUMBNAIL RESOLUTIONS TO CLIENT
2970
2910
CLIENT REQUESTS INTERACTION WITH ROI OF 3D IMAGE
PROGRESSIVELY TRANSMIT TO CLIENT IDENTIFIED ROI WITHIN 3D IMAGE
2980

FIG.29

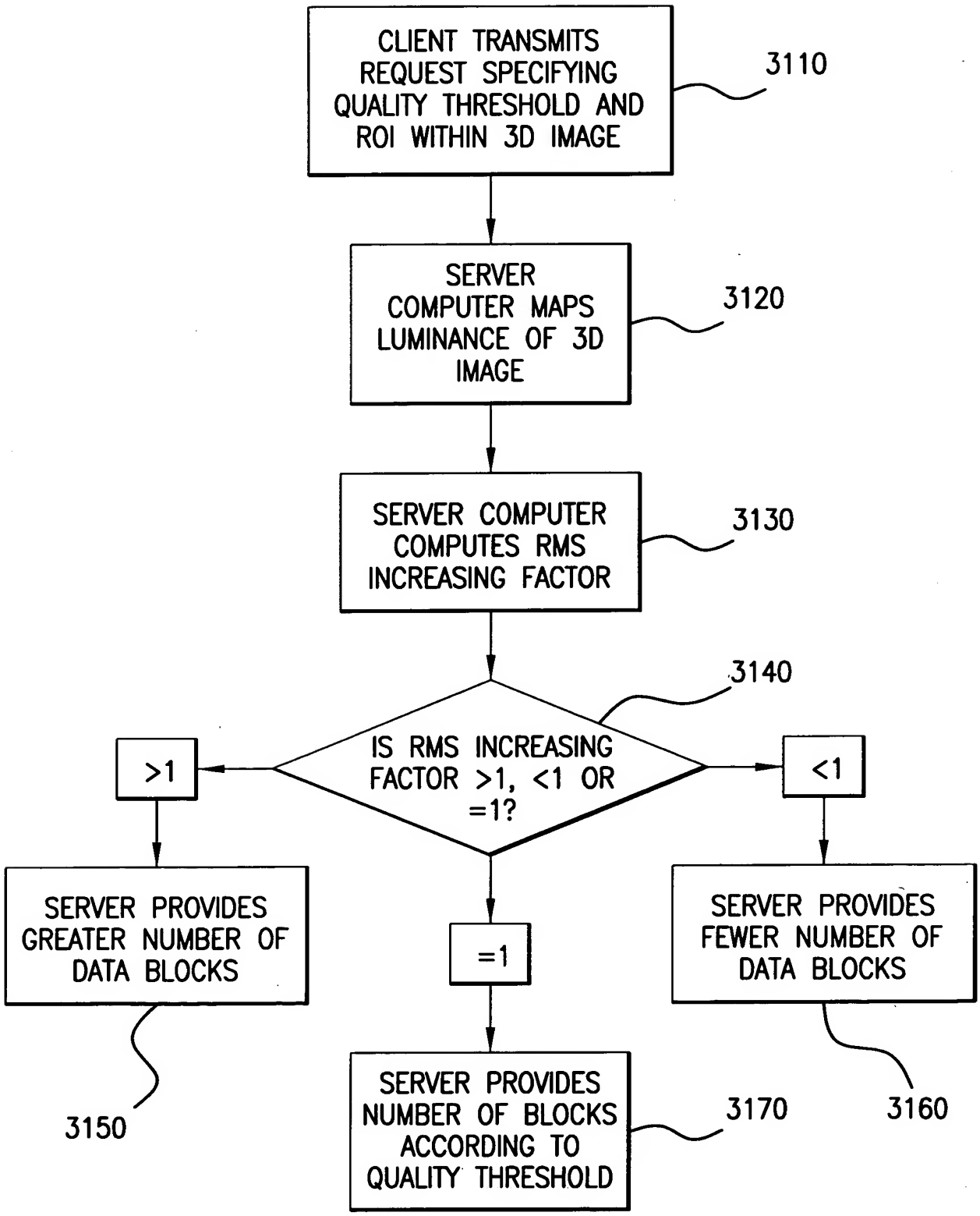

CLIENT TRANSMITS REQUEST SPECIFYING QUALITY THRESHOLD AND ROI WITHIN 3D IMAGE
3110
SERVER COMPUTER MAPS LUMINANCE OF 3D IMAGE
3120
SERVER COMPUTER COMPUTES RMS INCREASING FACTOR
3130
3140
IS RMS INCREASING FACTOR >1, <1 OR =1?
>1
<1
=1
SERVER PROVIDES GREATER NUMBER OF DATA BLOCKS
SERVER PROVIDES FEWER NUMBER OF DATA BLOCKS
SERVER PROVIDES NUMBER OF BLOCKS ACCORDING TO QUALITY THRESHOLD
3150
3160
3170

FIG.30